Universal Serial Bus
Device Class Definition
for Human Interface Devices

0.9d Draft Revision
February 1, 1996

Scope of this Revision

The 0.9c release candidate of this definition is intended for industry review.

Contributors

Alps	Mike Bergman
Cybernet	Tom Peurach
DEC	Tom Schmidt
Forte	Steve McGowan
KTC	Jody Crowe
LCS	Robert Dezmelyk
Logitech	Remy Zimmermann
Microsoft	Mike Van Flandern
ThrustMaster	Joe Rayhawk

Revision History

Rev	Date	Filename	Comments
0.9d	2/1/96	usb_hi9d.doc	Draft for industry review.

Please send comments via electronic mail to usbdevice@fsp008.fm.intel.com

Table of Contents

List of Tables

List of Figures

1. Introduction

The Universal Serial Bus (USB) is a communications architecture that gives a PC the ability to interconnect a variety of devices via a simple four-wire cable. The USB is actually a two-wire serial communication link that runs at 12 megabits (Mbs) per second. USB protocols can configure devices at startup or when they are plugged in at run time. These devices have been broken into various classes, such as display, communication, audio, mass storage, and human interface. A device class defines common functions and protocols for devices that serve similar functions.

1.1 Scope

This document describes the USB Human Interface Device Class (HID). Concepts from the USB Specification (particularly Chapter 9, "USB Device Framework") are used but not explained here. The HID Class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of class members are:

- Keyboards and pointing devices such as mice, trackballs or joysticks.
- Front-panel controls such as knobs, switches, buttons and sliders.
- Controls that might be found on devices such as a telephone, a VCR remote control, and game or simulation devices (for example, data gloves, throttles, steering wheels, and rudder pedals).
- Devices that may not require human interaction but provide data in a similar format to HID devices (for example, a bar-code reader, thermometer, or voltmeter).

Because human interface devices frequently include indicators, specialized displays, audio feedback, or even force or tactile feedback, this class also includes support for many types of output directed to the user.

1.2 Purpose

This document is intended to supplement the USB Specification and provide HID manufacturers with information necessary to build USB-compatible devices. It also specifies how the HID class driver should extract data from USB devices. The primary and underlying goals of this class definition are:

- Be as compact as possible to save device data space
- Allow easy extensions
- Allow the client to skip unknown information
- Be extensible and robust
- Support nesting and collections
- Be self-describing to allow generic clients

1.3 Related Documents

The following specifications and related documents are referenced by this class definition:

- *Universal Serial Bus Specification*, 1.0 final draft revision (also referred to as the *USB Specification*). In particular, see Chapter 9, "USB Device Framework."
- *The USB Class Specification for Legacy Software*
- HID Usage Tags (supplemental document, to be written)

1.4 Terms and Abbreviations

This section defines terms used throughout this document. For additional terms that pertain to the Universal Serial Bus, see Chapter 2, “Terms and Abbreviations,” in the USB Specification.

Array A series of data fields each containing an index that corresponds to an activated control. Banks of buttons or keys are reported in array entities.

Button bitmap A series of 1-bit fields, each representing the on/off state of a button. Buttons can be reported in either an array or a button bitmap.

Collection entity A meaningful grouping of Input, Output, and Feature entities. For example, mouse, keyboard, joystick, and pointer.

Control A destination for or source of report data. For example, mouse x axis, throttle, button, or LED.

Data entity An entity that adds fields to a report. For example, Input, Output and Feature entities are all data entities.

Designator Determines which body part is used for an entity or collection.

Entity A grouping of one or more controls with common attributes.

Feature entity Adds data field(s) to a Feature report. Feature controls affect the behavior of the device and are intended for use by device configuration utilities and not applications. For example, button repeat rate.

Input entity Adds data field(s) to an input report. Input controls are a source of data intended for applications. For example, x and y data.

Item A component of an entity descriptor that represents a piece of information about the device. The first byte of an item, called the item tag, identifies the kind of information an item provides.

Output entity Adds data field(s) to an output report. Output controls are a destination for data from applications. For example, LEDs.

Report A data structure returned by the device to the host (or vice versa) that is a representation of data entities. Some devices may have multiple report structures, each representing only a few entities. For example, a keyboard with an integrated pointing device could report key data independently of pointing data with the same interface.

Tag Part of an Entity descriptor that supplies information about the entity, such as its usage.

Transaction A device may send or receive a transaction every USB frame (1 ms). A transaction may be made up of multiple packets (token, data, handshake) but is limited in size to 8 bytes for low speed devices and 64 bytes for high speed devices.

Transfer A set of data that is meaningful to the device. Transfers may span one or more USB transactions.

Usage What entities are actually measuring as well as the vendor’s suggested use for specific entities.

Variable A data field containing a ranged value for a specific control. Any control reporting more than on/off needs to use a variable entity.

2. Management Overview

Information about USB devices is stored in device ROM, in segments called *descriptors*. The Device descriptor can identify a device as belonging to one of a finite number of classes (HID, Communication Device, Audio, Mass Storage, and so forth.) An HID class driver is responsible for retrieving and routing all data from USB devices identifying themselves as belonging to the HID class. This is accomplished by examining several descriptor sets that describe the device and the data it provides (HID, Entity, and String descriptors).

The HID Device descriptor indicates which other descriptors are present and their size. An Entity descriptor describes each piece of data that the device generates and what the data is actually measuring (for example, position or button press). This information can be used either to determine where to route input (for example, send it to mouse or joystick APIs) or to allow software to assign functionality to input (for example, use joystick input to position a tank). By examining all of the entities, collectively called the *Entity descriptor*, the HID class driver is able to determine the size and composition of data reports from the device. The String descriptor allows the device to associate a text string with particular entities (for example, button states).

3. Functional Characteristics

This section describes the functional characteristics of the HID class, its single subclass and its interfaces.

3.1 The HID Class

USB devices are segmented into classes. Devices in a given class have similar data transport requirements and consequently share a single class driver. For example, Audio devices require isochronous data pipes which are defined in the Audio class specification. HID devices have much simpler but different transport requirements identified in this specification. USB devices with data requirements outside the range of defined classes will need to provide their own drivers.

It is also possible to have devices composed of multiple classes. For example a telephone hand set might use features of the HID, Audio and Telephony classes. In these cases the class is specified in the Interface descriptor and not the Device descriptor.

3.2 Subclass

During the early development of the HID specification, subclasses were intended to be used to identify the specific protocols of different types of HID devices. For example, a Locator subclass would have identified the data report structure for a mouse, while Tablet, Game, Sim, or VR subclasses would have defined other unique protocols. While this mirrors the model currently in use by the industry (all devices use protocols defined by similar popular devices) it quickly became apparent that this approach was too restrictive. Devices would need to fit into narrowly defined subclasses and would not be able to provide any functionality beyond that supported by the subclass. The HID committee could not hope to create subclass protocols for all conceivable (and yet to be conceived) devices. In addition, many devices seemed determined to straddle multiple classifications (for example, keyboards with locators, or locators that provided keystrokes).

Consequently, the HID class does not use subclasses to define most protocols. Instead, an HID device identifies its data protocol and the type of data provided in the Entity descriptor. This descriptor is loaded and parsed by the HID class driver as soon as the device is detected. Protocols for existing devices as well as new, interesting devices are created by mixing flexible data types in the Entity descriptor.

Because the parser for the Entity descriptor represents a significant amount of code, a simpler method is needed to identify the device protocol for devices requiring BIOS support (boot devices). The HID class uses the Subclass parameter to indicate devices that support a predefined protocol for either mice or keyboards (that is the device can be used as a boot device). The boot protocol can be extended to include additional data not recognized by the BIOS, or the device may support a second preferred protocol for use by the HID driver. Boot entity descriptors are listed in Appendix B, “Boot Interface Descriptors.” For HID subclass and protocol codes, see Appendix E, “USB Standard Descriptors for HID Devices.”

3.3 Interfaces

An HID device communicates with the HID class driver via the Control pipe for USB control and class requests, and via an Interrupt pipe for sending data to the driver.

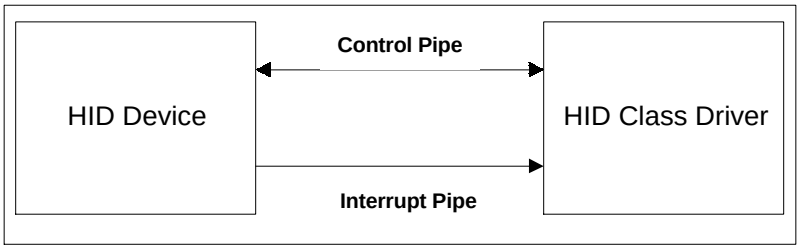

Figure 3-1: HID Data Flow

HID devices are interrupt-driven. Polled devices use the Control pipe for data transfer. Note that only the Interrupt pipe is described for the HID interface by an Endpoint descriptor. Endpoint 0 is a Control pipe always present in USB devices and therefore not described. In fact, several interfaces may share Endpoint 0. For details about the Control pipe, see the USB Specification.

Table 3-1: HID Pipes

Pipe	Description	Required?
Control	USB control and Class request codes	Y
Interrupt	Data in. Data from device	Y

3.4 Low speed versus high speed devices

This specification applies to both high speed and low speed HID devices.

Low speed devices have the following limitations:

- Transfer size is a maximum of 8 bytes per frame (1 ms).
- Only one low speed transaction is permitted per frame.
- Hubs cannot be low speed.
- Polling rate is currently defined at a maximum of every 10 ms.
- The size of a request should be minimized as much as possible (less than 8 bytes for common transfers) because low-speed control pipes have an average of one transaction every ten frames.

4. Operational Model

4.1 Descriptor Structure

The Device descriptor identifies the device class (HID), device subclass, vendor, product, and version (device). In the HID Class, subclass is currently used to identify boot devices. The HID Class driver identifies the exact type of device and features by examining additional descriptors (HID, Entity, and String).

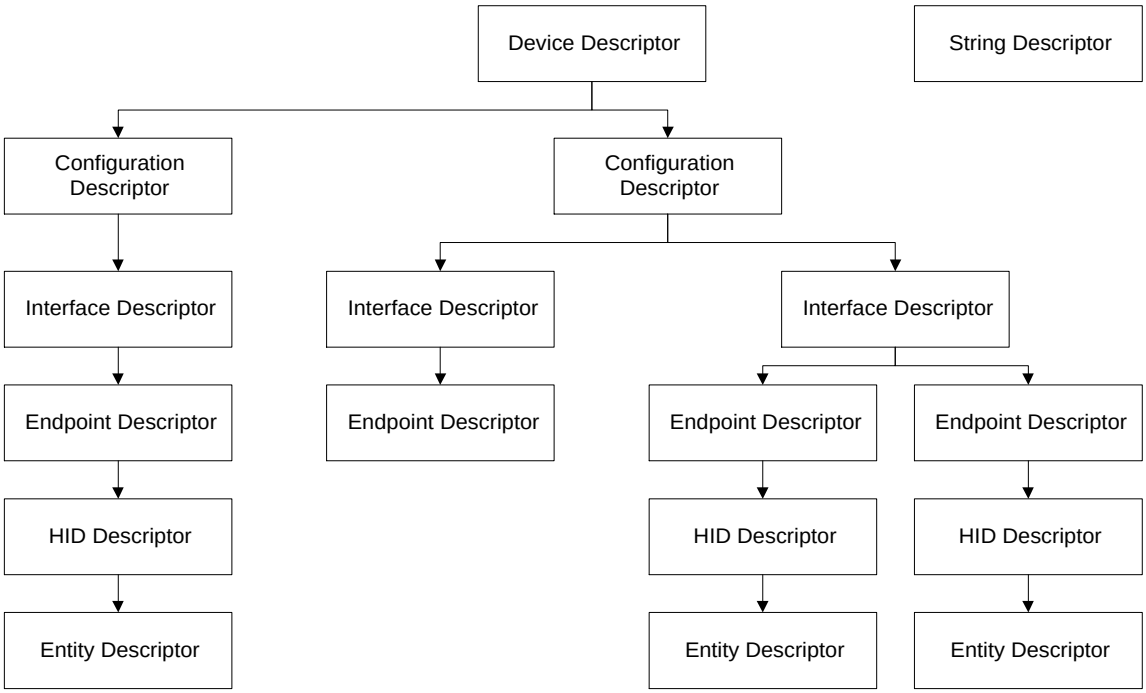

Figure 4-1: Descriptor Structure

4.2 Entities

An Entity descriptor is composed of individual entities. There are four types of entities: Input, Output, Feature and Collection.

- An Input or Output entity can refer to the data to or from a single control on a device such as a single axis or lever (variable data). Or, it can represent data from one or more push buttons or switches (array data).
- Feature entities describe device I/O not intended for consumption by applications (for example, key repeat rate).
- A Collection entity is a meaningful grouping of Input, Output and Feature entities (for example, mouse, keyboard, joystick, and pointer).

The Entity descriptor provides a description of the data provided by each control in a device. Each data entity (Input, Output, or Feature) identifies the size of data returned by a particular data entity, whether the data is absolute or relative, the minimum and maximum data values, and so forth. An Entity descriptor is the complete set of all data and collection entities for a device. By looking at an Entity descriptor alone, an application could tell how to handle incoming data as well as what the data could be used for.

4.3 Usages

Usage tags are part of the Entity descriptor and supply an application developer with information about what a control is actually measuring, as well as the vendor's suggested use for specific controls or groups of controls. While entities describe the format of the data (for example, three 8-bit fields), usage defines what should be done with the data (x, y and z input). This feature allows a vendor to insure that the user sees consistent function assignments to entities across applications.

Any entity can have multiple Usage tags. Each entity can represent one or more controls. Arrays and button bitmaps (variable entities with multiple 1-bit fields) actually represent several physical controls (buttons or keys). Each control may have attributes such as a usage assigned to them. For example, an array of four buttons could have a unique usage for each button.

The following is an example of how Entity descriptors and Usage tags could be associated to describe a keyboard with an integrated mouse.

Table 4-1: Entity and Usage Example

Entity	Usage Description	Number of Usage Tags
Collection (Application)		1
Collection (Application)	Keyboard	1
Input (Variable)	Modifier keys	8
Output (Variable)	LEDs	3
Input (Array)	Main keys	97
Collection (Application)	Mouse	1
Collection (Linked)	Pointer	1
Input (Variable)	X & Y	2
Input (Variable)	Buttons	2

4.4 Reports

Using USB terminology, a device may send or receive a transaction every USB frame (1 ms). A transaction may be made up of multiple packets (token, data, handshake) but is limited in size to 8 bytes for low speed devices and 64 bytes for high speed devices. A transfer is a set of data that is meaningful to the device. Transfers may span one or more USB transactions.

Most devices generate data reports (transfers) by returning a structure in which each data entity is sequentially represented. However, some devices may have multiple report structures in a single interface, each representing only a few entities. For example, a keyboard with an integrated pointing device could report key data independently from pointing data. Report tags can be used to indicate which entities are represented in each report structure. A Report tag assigns a 1-byte identification prefix to each report transfer. If no Report tags are present in the Entity descriptor, it can be assumed that only one Input, Output and Feature Report structure exists and they represent all of the device's data entities.

Note that only Input reports are sent via the Interrupt pipe. Feature and Output reports must be initiated by the host via the Control pipe.

If a device has multiple report structures, all data transfers start with a 1-byte identifier prefix that indicates which report structure applies to the transfer. This allows the class driver to distinguish incoming pointer data from keyboard data, by examining the transfer prefix.

4.5 Strings

A collection or data entity (or entity element) can have a particular label (string index) associated with it. Strings are optional.

The usage of an entity is not necessarily the same as a string associated with the entity. However, strings may be useful when a vendor-defined usage is required.

The String descriptor contains a list of text strings for the device. For details, see Appendix E, “USB Standard Descriptors for HID Devices.”

4.6 Format of Multibyte Numeric Values

Multibyte numeric values are represented in little-endian format, with the least significant byte at the lowest address. All integer values are signed values represented in 2’s complement format. Floating point values are not allowed.

The least significant bit in a value is stored in Bit 0, the next more significant in Bit 1 and so on up to the size of the value. For example, a long integer would be represented as shown in Table 5.1.

Table 4-1: Bit Representation of a Long Integer Value

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Bit 7 - 0							
Byte 1	Bit 15 - 8							
Byte 2	Bit 23 - 16							
Byte 3	Bit 31 - 24							

5. Descriptors

5.1 Standard Descriptors

The HID Device class uses the following USB standard descriptors:

- Device
- Configuration
- Interface
- Endpoint
- String

For details about these descriptors as defined for the HID Device class, see Appendix E, “USB Standard Descriptors for HID Devices.” For general information about USB standard descriptors, see Chapter 9, “USB Device Framework,” in the USB Specification.

5.2 Class-Specific Descriptors

The HID Device class uses the following class-specific descriptors:

- HID descriptor
- Entity descriptor

5.3 HID Descriptor

The HID descriptor identifies the length of subordinate descriptors for a device.

Table 5-1: HID Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptor	1	Constant	HID Descriptor type
2	bcdHID	2	BCD	HID Class Specification release number in binary-coded decimal (i.e. 2.10 is 0x210)
4	wEntityLength	2	Number	Total length of Entity descriptor
6	wDesignatorLength	2	Number	Total length of Designator descriptor

5.4 Entity Descriptor

The Entity descriptor is unlike other descriptors in that it is not simply a table of values. The length and content of an Entity descriptor vary depending on the number of data fields required for the device’s report(s). The Entity descriptor is made up of items, which provide information about the device. The first byte of an item, called the *item tag*, identifies the kind of information the item provides.

An entity is defined by an Entity item and further described by successive Attribute items. Attributes defined for a collection become the default attributes for all entities in that collection. An Entity descriptor must include at least the following items: Input or Output, Usage, Usage Page, Logical Minimum, Logical Maximum, Report Size and Report Count. All other items are optional.

Figure 5-1 is an example of items being used to define a mouse. Each line represents an entity or entity collection and additional items can provide more details about a given entity or collection. In this case, entities and collections are preceded with a Usage, Report Size or Report Count item (each line is a new item).

```
Usage Tag (Mouse),
Report Count Tag (0),
Collection Entity Tag: (Application),
    Usage Tag (Pointer),
    Collection Entity Tag (Linked),
        Usage Tag (X),
        Usage Tag (Y),
        Report Size Tag (8),
        Report Count Tag (2),
        Input Entity Tag: (Variable, Relative),
        Usage Tag (Button),
        Report Size Tag (1),
        Input Entity Tag: (Variable, Absolute),
    Report Count Tag (0),
    [End Collection Tag],
[End Collection Tag]
```

Figure 5-1: An Entity Descriptor that Defines a Mouse

5.4.1 Items

An item is piece of information about the device. The first byte of an item contains the item tag, which identifies the kind of information an item provides and indicates the size of any parameters associated with the tag.

Table 5-2: Item format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Item tag						Size: 0 = 0 bytes 1 = 1 byte 2 = 2 bytes 3 = 4 bytes	

The item tag identifies a particular item. A tag doesn't have an explicit parameter size associated with it. Instead, the value of a parameter determines the size requirement. That is, if a parameter value can be represented in one byte, the parameter can be specified as one byte, although this is not required.

If a large parameter is expected, it can still be abbreviated if all of its high-order bits are identical. The highest bit of the highest bit returned can be propagated. For example, a 32-bit parameter in which bytes 1, 2 and 3 are all 0 can be abbreviated as one byte if bit 7 of byte 0 is also 0.

For details about item tags, see sections 5.4.3, 5.4.4, and 5.4.4.1 in this document.

5.4.2 Item Parser

The HID class driver contains a parser used to analyze items found in the Entity descriptor. The parser extracts information from the descriptor in a linear fashion. The parser collects the state of each known item as it walks through the descriptor, and stores them in an item state table, which contains the state of individual items.

Some items are recognized as *terminating items*: whenever a terminating item is found, the contents of the item state table are moved. Examples of terminating items are Push, Pop, and entities.

When a Push item is encountered, the item state table is copied and placed on a stack for later retrieval. Pop replaces the item state table with the top table from the stack. For example:

Unit Tag (Meter), Unit Exp Tag (-3), Push Tag, Unit Exp Tag (0)

When the parser reaches the terminal item Push, it places the items defining units of millimeters (meters^{-3}) on the stack. The next item changes the item state table to units of meters (meters^{0}).

Whenever an Entity item is found, a new entity structure is allocated and initialized with the current item state table. In this way items set the default value for new entities. A device with several similar entities (for example, six axes) would need to define the items (extents, report size, units, and so forth) only once prior to the first entity.

Entities are associated with a collection by the order in which they are declared. A new collection starts when the parser reaches a collection item. Note that the parser associates an entity with a collection only when the entity is defined.

The parser is required to parse through the whole Entity descriptor to find all the entities. This is necessary in order to analyze reports sent by the device. For details, see section 7, “Report Protocol”.

From the parser’s point of view, an HID device looks like Figure 5-2.

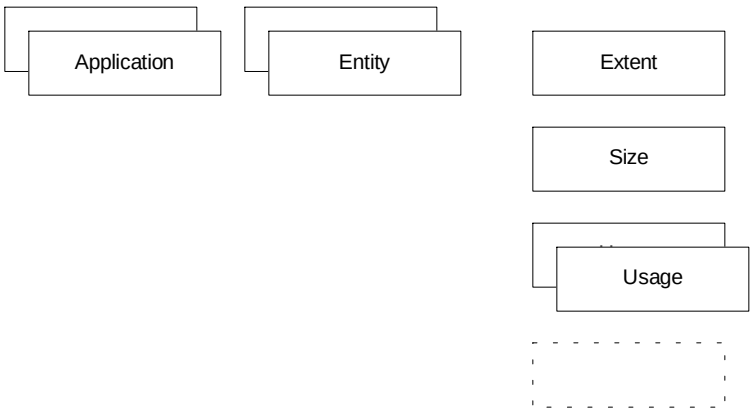

Figure 5-2: HID View from Parser

5.4.3 Entity Item Tags

Entities are created with entity item tags. Table 5-3 lists defined entity item tags.

Table 5-3: Entity Item Tags

Tag Code	Tag	Description and Parameters
100000xx	Input	Creates an Input entity. Valid parameters are: Bit 0 Array (0) / Variable (1) Bit 1 Absolute (0) / Relative (1) Bit 2 No Wrap (0) / Wrap (1) Bit 3 Linear (0) / Non Linear (1) Bit 4 Preferred State (0) / No Preferred (1) Bit 5 Null state (0) / No Null position (1) Bit 31-6 Reserved (0)
100001xx	Output	Creates an Output entity. Valid parameters are: Bit 0 Array (0) / Variable (1) Bit 1 Absolute (0) / Relative (1) Bit 2 No Wrap (0) / Wrap (1) Bit 3 Linear (0) / Non Linear (1) Bit 4 Preferred State (0) / No Preferred (1) Bit 5 Null state (0) / No Null position (1) Bit 6 Non Volatile (0) / Volatile (1) Bit 31-7 Reserved (0)
100010xx	Collection	Indicates the collection type. Entity collections are: 0x00 Linked (group of axes) 0x01 Application (mouse, keyboard) 0x04-0x7F Reserved 0x80-0xFF Vendor-defined
100011xx	Feature	Creates a Feature entity (See Output entity parameters).
100100xx	End Collection	Closes an Entity collection.
100101xx-111111xx	Reserved	Reserved for future Entity items

5.4.3.1 Input Entity

An input entity describes information about the data provided by one or more physical controls. An application can use this information to interpret the data provided by the device. All data in a single entity share an identical data format. The number of data fields in an entity can be determined by examining the Report Size and Report Count values. For example an entity with a Report Size of 8 bits and a Report Count of 3 has three 8-bit data fields. Input entities make input reports accessible via the control pipe with the Get/Set Input State command. Input Reports are also sent at the polling rate via the interrupt pipe.

Item parameters: Each bit in this byte represents a particular analog entity attribute. If the Input entity is an array, only the Variable/Array and Absolute/Relative attributes apply.

Variable/Array Whether the entity is variable or an array.

In variable entities, each field represents data from a physical control. The number of bits reserved for each control (field) is determined by Report Size/Report Count. For example, a bank of eight on/off switches could be reported as a 1-byte variable entity, on (1) or off (0) (Report Size =1, Report Count =8). Alternatively, a 1-byte variable entity could be used to represent the state of four three-position buttons, where the state of each button is represented by two bits (Report Size =2, Report Count =4). Or a 1-byte variable entity could represent the x position of a joystick ((Report Size =8, Report Count =1).

An array provides an alternate means for describing the data returned from a group of buttons. Arrays are more efficient, if less flexible than variable entities. Rather than returning a single bit for each button in the group, an array returns an index in each field that corresponds to the pressed button (like keyboard scan codes). An array field will return a 0 value when no controls in the array are pressed. Buttons or keys in an array that are simultaneously pressed need to be reported in multiple fields. Therefore the number of fields in an array entity (Report Count) dictates the maximum number of simultaneous controls that can be reported. A keyboard could report up to three simultaneous keys using an array with three 8-byte fields (Report Size =8, Report Count =3). Logical Min specifies the lowest index value returned by the array (other than 0) and Logical Max specifies the largest. The number of elements in the array can be deduced by examining the difference between Logical Min and Logical Max (number of elements = LOG MAX - LOG MIN + 1).

Absolute/Relative Indicates whether the data is absolute (based on a fixed origin) or relative (indicating the change in value from the last report). Mice usually provide relative data, while tablets usually provide absolute data.

Nonlinear The raw data from the device has been processed in some way, and no longer represents a linear relationship between what is measured and the data that is reported. Acceleration curves and joystick dead zones are examples of this kind of data. Sensitivity settings would affect the Units parameter, but the data would still be linear.

Wrap The data “rolls over” when reaching either the extreme high or low value. For example, a dial that can spin freely 360 degrees might output values from 0 to 10. If Wrap is indicated, the next value reported after passing the 10 position in the increasing direction would be 0.

Preferred State The control has a preferred state to which it will return when the user is not physically interacting with the control. Push buttons (as opposed to toggle buttons) and self-centering joysticks are examples.

Null State The control has a state in which it is not sending meaningful data. One possible use of the null state is for controls that require the user to physically interact with the control in order for it to report useful data. A Hat Switch is an example. When in a null state, the control will report a value outside of the specified Logical Min and Logical Max (the most negative value, such as -256 for an 8-bit value).

5.4.3.2 Output Entity

The Output entity is similar to an Input entity except it describes data sent to the device (for example, LED states). Output entities make Output reports accessible via the Control pipe with the Get/Set Output State command.

Item parameters: The Variable/Array, Absolute/Relative, Nonlinear, Wrap, and Null State parameters for an Output entity are identical to those parameters for an Input entity. The following parameters differ:

Preferred State Indicates that the control has a preferred state to which it will return shortly after being set by the host. For example, an LED may automatically turn itself off some time after being turned on by the host. The delay before returning to the preferred state is undefined.

Non-volatile/Volatile Non-volatile output can be changed only by the host, while volatile output can change with or without host interaction. If volatile output is absolute, issue a Set Output State command and set the value of any entity you don't want to change to a value outside of the specified Logical Min and Logical Max (the most negative value, such as -256 for an 8-bit value). This value indicates the Output entity's value should not be changed by the host.

5.4.3.3 Collection Entity

A collection entity identifies a relationship between two or more data entities. For example a mouse could be described as a collection of two to four data entities (x, y, button 1, button 2). Unlike data entities, collection entities do not generate data. However, like all entities, a Usage item may be associated with any collection (such as a mouse or throttle). Collection entities may be nested, and they are always optional.

The HID class supports two types of collection entities:

- **Application collection**. A group of entities that might be familiar to applications. It could also be used to identify entity groups serving different purposes in a single device. Common examples are a keyboard or mouse. a Keyboard with an integrated pointing device could be defined as two different application Collections. Data reports are usually (but not necessarily) associated with application collections (one report ID per application).
- **Linked collection**. A group of data entities describing data about a single object (like a point in space). A 3D device might have x, y and z position data entities in a single linked collection. Alternatively, a device could have two pointers with each x,y pair grouped in a separate linked collection. A joystick might report x and y, axes, and the buttons on the stick inside a linked collection, while buttons on the base could be reported outside the collection. A voltmeter might group voltage, resistance, and amperage from a set of probes into a linked collection.

5.4.3.4 Feature Entity

Feature entities describe device configuration information that can be sent to the device. While similar in function, Output and Feature entities differ in the following ways:

- Feature entities describe configuration options for the device and are usually set by a control panel application. Because they affect the behavior of a device (for example: button repeat rate, reset origin, and so forth), Feature entities are not usually visible to software applications. On the other hand, Output entities represent device output to the user (for example, LEDs, audio, tactile feedback, and so forth). Software applications are likely to set device Output entities.
- Feature entities may be attributes of other entities. For example, an Origin Reset Feature may apply to one or more position Input entities. Like Output entities, Feature entities make up Feature Reports accessible via the Control pipe with the Get/Set Feature State command.

Item parameters: Feature entity parameters are identical to parameters for Output entities.

5.4.3.5 End Collection

This tag defines the end of a collection. All entities between the Collection tag and End Collection tag are included in the collection. Collections may contain other nested collections.

Should origin reset be an output entity?

Should Retains position be part of a usage description?

5.4.4 Entity Attribute Item Tags

Entity attribute items describe rather than define an entity or collection. A new entity assumes the characteristics of the item state table. Entity attribute items following an entity can change the state table. Table 5-4 lists defined entity attribute item tags.

Table 5-4: Entity Attribute Item Tags

Tag Code	Tag	Description and Parameters
000001xx	Usage Page	Determine which Usage page is current. See Usage Reference Page.
000011xx	Logical Extent Minimum	Extent value in logical units.
000100xx	Logical Extent Maximum	Extent value in logical units.
000101xx	Physical Extent Minimum	Extent value in physical units.
000110xx	Physical Extent Maximum	Extent value in physical units.
001001xx	Unit Exponent	Value of the unit exponent in base 10.
001010xx	Unit	Unit values.
001011xx	Report Size	Size of each report field for the entity
001100xx	Report ID	Increments the Report ID. Report Rate indicates the maximum report rate for the new ID.
001101xx	Report Count	Number of report fields for the entity
010011xx	Push	Places a copy of the item state table on the stack.
010100xx	Pop	Replaces the item state table with the top structure from the stack
010101xx-011111xx	Reserved	

5.4.4.1 Usage Page

Since there are more than 256 usages, the Usage Page determines which set of usages are relevant. The Usage tag points to a particular usage on a given Usage Page.

For a list of Usage Page parameters, see Appendix A, “Usage Tags.”

5.4.4.2 Logical Extent Minimum/Maximum

This is the minimum and maximum value that a variable or array entity will report.

5.4.4.3 Physical Extent Minimum/Maximum

While logical extents bound the values returned by a device, physical extents give meaning to those bounds. For example, a thermometer might have logical extents of 0 and 999 but physical extents of 32 and 212 degrees.

The resolution can be calculated with the following formula:

$$\text{Resolution} = (\text{Logical Extent Maximum} - \text{Logical Extent Minimum}) / ((\text{Physical Extent Maximum} - \text{Physical Extent Minimum}) * (10^{\text{Unit Exponent}}))$$

For example, a 400-dpi mouse might have the items shown in the following table.

Table 5-5: Resolution Example for a 400-dpi Mouse

Item	Value
Logical Min	-127
Logical Max	127
Physical Min	-3175
Physical Max	3175
Exponent	-4
Unit	Inches

Item parameters: Minimum and maximum value for the physical extent of a variable entity.

5.4.4.4 Unit exponent

This is the base 10 exponent for units.

Item parameters: Exponent value. See Table 6.9, “Unit Systems.**Error! Reference source not found.**”

5.4.4.5 Unit

The Unit item qualifies values.

Table 5-6: Unit Parameters

Nibble 3	Nibble 2	Nibble 1	Nibble 0
Time	Mass	Length	System

Table 5-7: Unit Systems

Code	System	Length	Mass	Time	Exponent
0x0	None	None	None	None	0
0x1	SI Linear	Meter	Gram	Seconds	0
0x2	SI Rotation	Radians	Gram	Seconds	1
0x3	English Linear	Inch	Slug	Seconds	2
0x4	English Rotation	Degrees	Slug	Seconds	3
0x5	Electric	Resistance	Ampere	Seconds	4
0x6	Temperature	Kelvin		Seconds	5
0x7	Luminosity	Candela		Seconds	6
0x8	TBD				7
0x9	Vendor-defined				-8
0xA	TBD				-6
0xB	TBD				-5
0xC	TBD				-4
0xD	TBD				-3
0xE	TBD				-2
0xF	TBD				-1

All complex units can be composed from length, mass and time. For example energy (joules) can be represented as [grams][meters^{2}][second^{-2}] (the unit exponent would be 3 because grams are used instead of Kg to compose the Unit). The parameters of some common units are shown in the following table:

Table 5-8: Unit Examples

Unit	Nibble 3	Nibble 2	Nibble 1	Nibble 0	Code
Distance	0	0	1	1	x0011
Mass	0	1	0	1	x0101
Time	1	0	0	1	x1001
Velocity	-1	0	1	1	xF011
Momentum	-1	1	1	1	xF111
Acceleration	-2	0	1	1	xE011
Force	-2	1	1	1	xE111
Energy	-2	1	2	1	xE121
Angular Acceleration	-2	0	1	2	xE012

5.4.4.6 Report size

This describes the report size in bits of each field in the entity. This allows the parser to build an entity map for the report handler to use.

For more information, see section 7, “Report Protocol.”

Item parameters: Size of the report fields in bits. This value is an unsigned integer (unlike nearly every other value in this document).

5.4.4.7 Report ID

If a Report ID tag is used anywhere in the Entity descriptors, all data reports for the device are preceded by a single byte ID field. All entities preceding the first Report ID tag are included in a report prefixed by 0x00. All entities preceding the second but following the first Report ID tag are included in a report prefixed by 0x01, and so forth. For more information, see section 7, “Report Protocol.”

Item parameters: Report rate in reports per seconds

This is the maximum report rate the device will use for this particular report when its state changes. Note that this number cannot exceed the device report rate found in the Device descriptor. For pointing devices, this must be high enough to avoid introducing too much latency in the device state change and the screen update. For array entities like keyboards, the report rate must be high enough so that the software can generate an acceptable repeat rate for the keys. While the host does not usually need to know the report rate of individual reports, it may be useful in the case of time-dependent data, such as pointer data prior to adding an acceleration curve.

5.4.4.8 Report Count

Report Count determines how many fields are included in the report for this particular entity (and consequently how many bits are added to the report).

In the case of an array, Report Count determines the maximum number of controls that may be included in the report and consequently the number of key or buttons that may simultaneously be pressed as well as the size of each element. For example, an array supporting up to three simultaneous key presses, where each field is one byte, would look like this: Report Size = 8, Report Count = 3.

In the case of a variable entity, the Report Count specifies how many controls are included in the report. For example, eight buttons could look like this: Report Size = 1, Report Count = 8.

For more information, see section 7, “Report Protocol.”

Item parameters: The number of fields in the report.

5.4.4.9 Push

When a Push item is encountered, the item state table is copied and placed on a stack for later retrieval.

Item parameters: N/A

5.4.4.10 Pop

The Pop item replaces the item state table with the top structure from the stack.

Item parameters: N/A

5.4.5 Control Attribute Item Tags

Control attribute items define characteristics of controls. These items do not carry over to the next entity. If the entity contains more than one control, it may be preceded by several similar control attributes. For example, an entity may have several usage tags associated with it, one for each control.

While control attributes do not carry over to the next entity, they may apply to more than one control within a single entity. For example, if an entity defining five controls is preceded by three usage tags, the three usages would be assigned sequentially to the first three controls, and the third usage would also be assigned to the fourth and fifth controls. If an entity has no controls (Report Count = 0), the control attributes apply to the entity (usually a collection entity). Table 5-9 lists defined control attribute items.

Table 5-9: Control Attribute Item Tags

Tag Code	Tag	Description and Parameters
000000xx	Usage	Usage index for an entity usage. See Appendix A.
000111xx	Usage Min	Defines the starting usage associated with an array or bitmap.
001000xx	Usage Max	Defines the ending usage associated with an array or bitmap.
001110xx	Designator Index	Determines the body part used for a control. Index points to a designator in the Designator descriptor
001111xx	Designator Min	Defines the starting designator associated with an array or bitmap
<<code>>	Designator Max	Defines the ending designator associated with an array or bitmap
010000xx	String Index	String index for a String descriptor. See Appendix E.
010001xx	String Min	Defines the starting string associated with an array or bitmap
010010xx	String Max	Defines the ending string associated with an array or bitmap
010101xx-011111xx	Reserved	

5.4.5.1 Usage

The Usage tag represents a suggested usage for the entity or collection. In the case where an entity represents multiple controls, a Usage tag may suggest a usage for every variable or element in an array. To assign unique usages to every control in a single entity, simply specify each Usage tag sequentially or use Usage Min/Usage Max).

Note: It is important that Usage be used properly. While very specific usages exist (landing gear, bicycle wheel, etc.) those usages are intended to identify very narrow-purpose devices. A joystick with generic buttons should never assign an application-specific usage to any button. Instead, it should assign a generic usage such as “Button.” However, an exercise bicycle or the cockpit of a flight simulator may want to narrowly define the function of each of its data sources.

It’s also important to remember that Usage items convey information about the intended use for the data and may not correspond to what is actually being measured. For example, a joystick would have X & Y Usages associated with its axis data, and not Usages Rx & Ry.

If the same usage is assigned to several controls, the order in which the controls appear in the report indicates their priority. For example, the first button is the primary button. In the case of arrays, controls are prioritized in ascending index order.

For a list of Usage parameters, see Appendix A, “Usage Tags.”

5.4.5.2 Usage Minimum/Maximum

Because button bitmaps and arrays can represent multiple buttons or switches with a single entity, it may be useful to assign multiple usages to an entity. Usage Min specifies the usage to be associated with the first unassociated control in the array or bitmap. Usage Max specifies the end of the range of usage values to be associated with entity elements. The following example illustrates how this could be used for an 84-key keyboard with three new Windows keys.

Table 5-10 Example of an Entity with Usage Min and Usage Max

Tags	Result
Report Count (1)	Add 1 field to the report
Report size (8)	Field is 1 byte
Log Min (1)	Returns values from 1...
Log Max (87)	... to 87 (and 0)
Usage Page (keycode page)	Assigns IBM keyboard usages
Usage Min (first key)(0x01)	Assigns 84-key usages
Usage Max (last key)(0x54)	
Usage Min (LWin)	Assigns Win key usages
Usage Max (RWin)(0x6F)	
Usage (App)	
Input: (Array, Absolute)	Creates the array

Item parameters: The minimum and maximum value that the device will report.

5.4.5.3 Designator Index

Determines the body part used for a control. The index points to a designator in the Designator descriptor.

Item parameters: Index of the Designator descriptor.

5.4.5.4 Designator Min

Specifies the first designator associated with an array or bitmap.

Item parameters: Index of the first Designator descriptor in the group.

5.4.5.5 Designator Max

Specifies the ending designator associated with an array or bitmap.

Item parameters: Index of the last Designator descriptor in the group.

5.4.5.6 String Index

This tag allows a string to be associated with a particular entity or control.

Item parameters: Index of the String descriptor.

5.4.5.7 String Min

Specifies the first String Index when assigning a group of sequential strings to controls in an array or bitmap.

Item parameters: Index of the first String descriptor in the group.

5.4.5.8 String Max

Specifies the last String Index when assigning a group of sequential strings to controls in an array or bitmap.

Item parameters: Index of the last String descriptor in the group.

6. Requests

6.1 Standard Requests

Standard USB requests such as Get Descriptor are described in the USB Specification. For details, see Chapter 9, “USB Device Class Framework.”

6.2 Class-Specific Requests

Class-specific requests allow the host to inquire about the capabilities and state of a device and to set the state of output and feature entities. These transaction are done over the default pipe and therefore follow the format of default pipe requests as defined in the USB Specification.

Table 6-1: Request Format

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bitmap	Characteristics of request: D7 Data transfer direction 0 = Host to device 1 = Device to host D6..5 Type 1 = Class D4..0 Recipient 2 = Endpoint
1	bRequest	1	Value	Specific request. See Table 6-2.
2	wValue	2	Number	Word-size field that varies according to the request.
4	wIndex	2	Index or offset	Word-size field that varies according to the request.
6	wLength	2	Count	Number of bytes to transfer in the data phase.

The following table defines the values of class-specific requests:

Table 6-2: Class Request Values

Value	Description
0x00	GET_HID_DESCRIPTOR
0x01	SET_HID_DESCRIPTOR[1]
0x02	GET_ENTITIES_DESCRIPTOR
0x03	SET_ENTITY_DESCRIPTOR[1]
0x04	GET_INPUT_STATE
0x05	SET_INPUT_STATE[1]
0x06	GET_OUTPUT_STATE
0x07	SET_OUTPUT_STATE
0x08	GET_FEATURE_STATE[1]
0x09	SET_FEATURE_STATE[1]
0x0A	IDLE_REPORT
0x0B	CHANGE_PROTOCOL

[1]These requests are optional and may not be supported by all devices.

6.2.1 Get HID Descriptor

This request returns the HID descriptor for the device. This is typically the first class-specific request sent to the device, because it returns descriptor size information for subsequent requests.

Table 6-3: Get HID Descriptor Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_HID_DESCRIPTOR	Zero	Interface	Descriptor Length	HID Descriptor

Table 6-4: Set HID Descriptor Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_HID_DESCRIPTOR	Zero	Interface	Descriptor Length	HID Descriptor

6.2.2 Get Entities Descriptor

The device replies to this request with the Entities descriptor.

Table 6-5: Get Entity Descriptor Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_ENTITY_DESCRIPTOR	Zero	Interface	Descriptor Length	Entity Descriptor

Table 6-6: Set Entity Descriptor Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_ENTITY_DESCRIPTOR	Zero	Interface	Descriptor Length	Entity Descriptor

Get/Set DesignatorDescriptor commands are also needed. However, development of these commands is being postponed until the USB core team finalizes a proposal for handling device class descriptors. This proposal may change the Get/Set HID and Entity descriptor commands.

6.2.3 Get Input State

This request allows the host to get the state of input entities. This is particularly important at initialization time for absolute entities. This command is not intended to be used for polling the device state on a regular basis. The input state reply has the same format as the reports from the device.

Table 6-7: Get Input State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_INPUT_STATE	Zero Report ID	Interface	Descriptor Length	Input Report

Table 6-8: Set Input State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_INPUT_STATE	Zero Report ID	Interface	Descriptor Length	Input Report

6.2.4 Get Output State

This request allows the host to get the current state of an application output entities. This command should be used only at initialization time; the host should model the state of the application under normal operations.

Table 6-9: Get Output State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_OUTPUT_STATE	Zero Report ID	Interface	Descriptor Length	Output Report

6.2.5 Set Output State

This request allows the hose to set the state of the Output entities of a report.

Table 6-10: Set Output State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_OUTPUT_STATE	Zero Report ID	Interface	Descriptor Length	Output Report

6.2.6 Get Feature State

This request allows the host to get the state of the Feature entities of a report.

Table 6-11: Get Feature State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_FEATURE_STATE	Zero Report ID	Interface	Descriptor Length	Feature Report

6.2.7 Set Feature State

This request allows the host to set the state of the Feature entities of a report.

Table 6-12: Set Feature State Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_FEATURE_STATE	Zero Report ID	Interface	Descriptor Length	Feature Report

6.2.8 Idle Report

This request silences reports on the interrupt pipe until a new event occurs or the specified amount of time passes.

This command still needs to be specified. How is time specified? How can a particular report be turned off without affecting other reports? TBD

Table 6-13: Idle Report Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	IDLE_REPORT	TBD	Interface	Descriptor Length	Input Report

6.2.9 Change Protocol

This request is supported by devices in the boot subclass. This request switches between the boot protocol and the entity protocol (or vice versa). The **wValue** field dictates which protocol should be used.

Table 6-14: Disable Boot Protocol Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	CHANGE_PROTOCOL	0 Boot 1 Entity	Interface	Descriptor Length	N/A

7. Report Protocol

7.1 Report types

Reports contain data from one or more entities. Data transfers are sent from the device to the host through the Interrupt pipe in form of reports. Reports may also be requested (polled) and sent through the Control pipe. A report contains the state of all the entities (Input, Output or Feature) belonging to a particular Report ID. The client is responsible for extracting the individual entities from the report based on the Entity descriptor.

All of the entities' values are packed on bit boundaries in the report (no byte or nibble alignment). However, entities reporting null or constant values may be used to byte-align values if desired, or the Report Size may be made larger than needed for some fields simply to extend them to a byte boundary.

The bit length of an entity's data is obtained through the Entity descriptor (Report Size * Report Count). Entity data is ordered just as entities are ordered in the Entity descriptor. If a Report ID tag was used in the Entity descriptor, all reports include a single byte ID prefix. If the Report ID tag was not used, all values are returned in a single report and a prefix ID is not included in that report.

7.2 Report Format for Standard Entities

The report format is composed of an 8-bit report identifier followed by the entities belonging to this report.

Table 7-1: Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Report ID							
Byte 1 - n	Entities							

7.2.1 Report ID

The Report ID field is 8 bits in length. The value of the Report ID field is based on the Entity descriptor, starting with 0 and incrementing by 1 for each Report ID tag. If no Report ID tags are used in the Entity descriptor, there is only one report and the Report ID field is omitted.

7.2.2 Entities

The Entities fields are variable-length fields that report the state of an entity.

7.3 Report Format for Array Entities

Each button in an array reports an assigned number called an array index. This can be translated into a key code by looking up the array elements Usage Page and Usage. When any button transitions between open and closed, the entire list of indices for buttons currently closed in the array is transmitted to the host. Since only one array element can be reported in each array field, modifier keys should be reported as bitmap (1-bit variable) data. The following example shows the reports generated by a user typing Alt-Ctrl-Del, using a bitmap for the modifiers and a single array for all other keys.

Table 7-2: Example of a Report Generated by Alt-Ctrl-Del

Transition	Modifier Byte	Array Byte
Alt down	Alt	Null
Ctrl down	Alt Ctrl	Null
Del down	Alt Ctrl	Del
Del up	Alt Ctrl	Null
Ctrl up	Alt	Null
Alt up	<empty list>	Null

For a list of standard keyboard key codes, see Appendix A, “Usage Tags.”

If there are multiple reports for this device, all reports would be preceded by the Report ID.

Table 7-3: Example of a Report with a Report ID

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Report ID							
Byte 1	Bitmap							
Byte 2 - n	Array Index							

If a set of keys or buttons cannot be mutually exclusive, they must be represented either as a bitmap or as multiple arrays. For example, function keys on a 101-key keyboard are sometimes used as modifier keys (For example, F1 A). In this case, at least two array fields should be reported in an array entity (Report Count = 2).

7.4 Report Constraints

The following constraints apply to reports and to the report handler:

- An entity field cannot span more than four bytes in a report. For example, a 32-bit entity must start on a byte boundary to satisfy this condition.
- More than one report can be present in one USB transfer. For example, an 8-byte USB transfer could contain two Input reports.
- A report might span one or more USB transactions. For example, an application that has 10-byte reports will span at least two USB transactions in a low speed device.
- A report is always byte-aligned. If required, reports are padded with bits (0) until the next byte boundary is reached.

7.5 Report Example

The following Entity descriptor defines an entity with an Input report.

```
Usage Pg (Pg#), Usage (Mouse), Report Count (0),
Collection: (Application),
    Usage Pg (Pg#), Usage (Pointer),
    Collection: (Linked),
        Report ID, Usage Pg (Pg# Pointer), Usage (X), Usage (Y),
        Log Min (-127), Log Max (127), Report Size (8), Report Count (2),
        Input: (Variable, Relative),
        Log Min (0), Log Max (1), Report Count (1), Report Size (2),
        Usage Pg (Pg# for Buttons), Usage (Button),
        Input: (Variable, Absolute),
    [End Collection]
[End Collection]
```

Figure 7-1: Entity Descriptor for an Entity with an Input Report

The Input report structure for the above device would look as follows:

Table 7-4: Example of an Input Report Structure

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Report ID (0)							
Byte 1	Entity 0							
	Entity 1							
Byte n	0						Entity 3	Entity 2

Let’s use the keyboard with integrated pointing device example to demonstrate how to use two reports for a device with just one interface.

Table 7-5: Example of Two Reports for a Device with One Interface

Entity	Usage(s)	Report ID
Collection (Application)	Keyboard	
Input (Variable, Absolute)	Modifier Keys	00
Output (Variable, Absolute)	LEDs	00
Input (Array, Absolute)	Main Keys	00
End Collection		
Collection (Application)	Mouse	
Collection (Linked)	Pointer	
Input (Variable, Relative)	X, Y	01
Input (Variable, Absolute)	Button	01

Note: Only data entities (not collection entities) present data in a report. Also this example demonstrates the use of multiple reports, however this interface would not be exceptable for a boot device.

7.6 Status

7.6.1 Error

TBD

Appendix A Usage Tags

See the supplemental document for usage tags, including key codes for keyboards.

More information will be provided in this document in a future release.A1

Appendix B Boot Interface Descriptors

The HID Subclass 1 defines two descriptors for boot devices. Devices may append additional data to these boot reports, but the first n bytes of all reports must be identical in order for the data to be correctly interpreted by the BIOS. The BIOS will ignore any extensions to reports. These descriptors describe reports that the BIOS expects to see. However, since the BIOS does not actually read the Entity descriptors, these descriptors do not have to be hardcoded into the device. Instead, descriptors that describe the device reports in a USB-aware OS should be included (these may or may not be the same). When the HID class driver is loaded, it will issue a Change Protocol, changing from the boot protocol to the entity protocol after reading the boot interface's Entity descriptor.

B.1 Protocol 1 (Keyboard)

The following illustration represents an Entity descriptor for a boot interface for a keyboard.

```
Usage Pg (Pg#), Usage (Keyboard), Report Count (0),
Collection: (Application),
    Usage Pg (Pg# Scan Codes); Usage Min (#), Usage Max (#),
    Log Min (0), Log Max (1), Report Size (1), Report Count (16),
    Input: (Variable,Absolute),
    Report Count (3), Report Size (1), Usage Pg (Pg# for LEDs),
    Usage Min (#), Usage Max (#),
    Output: (Variable, Absolute)
    Report Count (6), Report Size (8), Log Min (0), Log Max(96),
    Usage Pg (Pg# Scan Codes), Usage Min (#),,,Usage Max (#),
    Input: (Array),
[End Collection]
```

Figure B-1: Boot Interface Descriptor for a Keyboard

Table B-1: Keyboard Report

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Modifier keys							
Byte 1	Reserved							
Byte 2	Keycode 1							
Byte 3	Keycode 2							
Byte 4	Keycode 3							
Byte 5	Keycode 4							
Byte 6	Keycode 5							
Byte 7	Keycode 6							

B.2 Protocol 2 (Mouse)

The following illustration represents an Entity descriptor for a boot interface for a mouse.

```
Usage Pg (Pg#), Usage (Mouse), Report Count (0)
Collection: (Application),
    Usage Pg (Pg#), Usage (Pointer),
    Collection (Linked),
        Usage Pg (Pg# for Buttons), Usage (Button),
        Log Min (0), Log Max (1), Report Count (8), Report Size (1),
        Input: (Variable, Absolute)
        Usage (X), Usage (Y),
        Log Min (-127), Log Max (127), Report Size (8), Report Count (2),
        Input: (Variable, Relative),
    [End Collection],
[End Collection]
```

Figure B-2: Boot Interface Descriptor for a Mouse

Table B.2 Mouse Report

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Device specific					Button 3	Button 2	Button 1
Byte 1	X displacement							
Byte 2	Y displacement							
Byte 3-n	Device specific							

7.7 B3

Appendix C Keyboard Implementation

The following are design requirements for USB keyboards:

1. Most keys should be reported in Input (Array, Absolute) entities. Reports will contain a list of keys currently pressed and not make/break codes (relative data).
2. The keyboard should send data reports when polled, even when there are no new key events.
3. The keyboard should support the Idle command.
4. The keyboard should report a phantom state in all array fields when the number of keys pressed exceeds the Report Count. The limit is three non-modifier keys when using the keyboard descriptor in Appendix B. Additionally, a keyboard may report the phantom condition when an invalid or unrecognizable combination of keys is pressed.
5. The order of keycodes in array fields has no significance. Order determination is done by the host software comparing the contents of the previous report to the current report. If two or more keys are reported in one report, their order is indeterminate. Keyboards may buffer events that would have otherwise resulted in multiple event is a single report.
6. “Repeat Rate” and “Delay before First Repeat” are implemented by the host and not in the keyboard (this means the BIOS in legacy mode). The host may use the device report rate and the number of reports to determine how long a key is being held down. Alternatively the host may use its own clock or the idle command for the timing of these features. For an example, see Table C.1.
7. Synchronization between LED states and Caps, Num or Scroll Lock events is maintained by the host and NOT the keyboard. If using the keyboard descriptor in Appendix B, LED states are set by sending a 3-bit report to the keyboard via a Set Output State command.
8. For boot keyboards, the reported index for a given key must be the same value as the key usage for that key. This is required because the BIOS will not read the Entity descriptor. It is recommended (but not required) that non-legacy protocols also try to maintain a one-to-one correspondence between indices and usage tags where possible.
9. Boot keyboards must support the boot protocol and the Change Protocol command. Boot keyboards may support an alternative protocol (specified in the entity descriptor) for use in USB-aware operating environments.

Table C-1: Example of keyboard output (4-byte reports)

Key Event	Modifiers Byte	Array	Array	Array	Comment
none	00000000B	00H	00H	00H	
RShift down	01000000	00	00	00	
None	01000000	00	00	00	Report current key state even when no new key events
A down	01000000	2E	00	00	
X down	01000000	2E	3D	00	
B down	01000000	2E	40	3D	Report order is arbitrary and does not reflect order of events
Q down	01000000	FF	FF	FF	Phantom state. Four Array keys pressed. Modifiers still reported.
A up	01000000	20	40	3D	
B & Q up	01000000	3D	00	00	Multiple events in one report. Event order is indeterminate.
none	01000000	3D	00	00	
RShift up	00000000	3D	00	00	
X up	00000000	00	00	00	

Note: This example uses a 4-byte report so that the phantom condition can be more easily demonstrated. Most keyboards should have 8 or more bytes in their reports.

Appendix D Example Entity Descriptors

The following are example descriptors for common devices. These examples are provided only to assist in understanding this specification and are not intended as definitive solutions.

D.1 Example Joystick Descriptor

```
Usage Pg (Pg#), Usage (Joystick), Report Count (0)
Collection: (Application),
    Usage Pg (Pg#), Usage (Pointer),
    Collection (Linked),
        Usage (X), Usage (Y), Log Min (-127), Log Max (127),
        Report Size (8), Report Count (2), Push
        Input: (Variable, Absolute),
        Usage (Hatswitch). Log Min (0), Log Max (3), Phy Min (0),
        Phy Max (270), Unit (Degrees), Report Count (1), Report Size (4),
        Input: (Variable, Absolute, Null State)
        Log Min (0), Log Max (1), Report Count (2), Report Size (1),
        Usage Pg (Pg# for Buttons), Usage (Button), Unit (None),
        Input: (Variable, Absolute)
    [End Collection],
    Usage (Button)
    Input: (Variable, Absolute)
    Usage (Throttle), Log Min (-127), Log Max (127),
    Report Size (8), Report Count (1),
    Input: (Variable, Absolute),
[End Collection]
```

Figure D-1: An Entity Descriptor that Defines a Joystick

Table D-1: Example Joystick Report

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	X position							
Byte 1	Y position							
Byte 2	Btn 4	Btn 3	Btn 2	Btn 1	Hatswitch			
Byte 3	Throttle							

Note: While the Hatswitch entity only requires 3 bits, it is allocated 4 bits in the report. This conveniently byte-aligns the remainder of the report.

Appendix E USB Standard Descriptors For HID Devices

E.1 Device Descriptor

Table E-1: HID Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x18	Size of this descriptor in bytes
1	bDescriptorType	1	0x01	DEVICE descriptor type
2	bcdUSB	2	0x????	USB Specification Release
4	bDeviceClass	1	0x00	Class code (assigned by USB).
5	bDeviceSubClass	1	0x00	Subclass code. 0 No subclass 1 Boot Interface subclass
6	bDeviceProtocol	1	0x00	Protocol code 0 None 1 Keyboard 2 Mouse
7	bMaxPacketSize0	1	0x08	Maximum packet size for endpoint zero (only 8, 16, 32, or 64 are valid)
8	idVendor	2	0x????	Vendor ID (assigned by USB)
10	idProduct	2	0x????	Product ID (assigned by manufacturer)
12	bcdDevice	2	0x????	Device release number in Binary-Coded Decimal
14	iManufacturer	1	0x01	Index of string descriptor describing manufacturer
15	iProduct	1	0x02	Index of string descriptor describing product
16	iSerialNumber	1	0x03	Index of string descriptor describing the device’s serial number
17	bNumConfigurations	1	0x01	Number of possible configurations

E.2 Configuration

Table E-2: Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor in bytes
1	bDescriptorType	1	0x02	CONFIGURATION (USB Defined)
2	wTotalLength	2	0x????	Total length of data returned for this configuration. Includes the combined length of all descriptors (configuration, interface, endpoint and class or vendor specific) returned for this configuration. Dependent on string descriptor size. This value does not include HID class descriptors.
4	bNumInterfaces	1	0x01	Number of interfaces supported by this configuration
5	bConfigurationValue	1	0x00	Value to use as an argument to Set Configuration to select this configuration
6	iConfiguration	1	0x04	Index of string descriptor describing this configuration
7	bmAttributes	1	10100000B	Configuration characteristics D7 Bus Powered D6 Self Powered D5 Remote Wakeup D4..0 Reserved (reset to 0)
8	MaxPower	1	0x??	Maximum power consumption of USB device from bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e. 50 = 100 mA).

E.3 Interface

Table E-3: Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor in bytes
1	bDescriptorType	1	0x04	INTERFACE Descriptor Type (USB Defined)
2	bInterfaceNumber	1	0x00	Number of interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	0x00	Value used to select alternate setting for the interface identified in the prior field
4	bNumEndpoints	1	0x00	Number of endpoints used by this interface (excluding endpoint zero). If this value is zero, this interface only uses endpoint zero.
5	bInterfaceClass	1	0x03	Class code (HID code assigned by USB)
6	bInterfaceSubClass	1	0x00	Subclass code (assigned by USB). These codes are qualified by the value of the *bInterfaceClass* field.
7	bInterfaceProtocol	1	0x01	Protocol code (assigned by USB).
8	iInterface	1	0x05	Index of string descriptor describing this interface

E.4 Endpoint Descriptor

Table E-4: Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor in bytes
1	bDescriptorType	1	0x05	ENDPOINT Descriptor Type (USB defined)
2	bEndpointAddress	1	10000001B	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: Bit 0..3: The endpoint number Bit 4..6: Reserved, reset to zero Bit 7: Direction, ignored for Control endpoints: 0 OUT endpoint 1 IN endpoint
3	bmAttributes	1	00000011B	This field describes the endpoint's attributes when it is configured using the bConfigurationValue. Bit 0..1: Transfer type: 00 Control 01 Isochronous 10 Bulk 11 Interrupt All other bits are reserved
4	wMaxPacketSize	2	0x08	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. For interrupt endpoints, this value is used to reserve the bus time in the schedule, required for the per frame data payloads. Smaller data payloads may be sent, but will terminate the transfer and thus require intervention to restart.
6	wSampleSize	2	0x00	This field is used for isochronous endpoints only. For all other types of endpoints, this field must be reset to zero.
8	bInterval	1	0x10	Interval for polling endpoint for data transfers. Expressed in milliseconds.

E.5 String Descriptor

Table E-5: String Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	STRING Descriptor Type
2	bString	N	Number	Array of LangID codes

E.6 Strings

Table E-6: Strings

bLength	Descriptor Type	Unicode-Encoded String	Index
0x??	0x03	Array of LangID codes	0
0x??	0x03	Manufacturer	1
0x??	0x03	Product	2
0x??	0x03	Device serial number	3
0x??	0x03	Configuration	4
0x??	0x03	Interface	5
0x??	0x03	Entity strings	6+